GOP LEVEL PARALLELISM IMPLEMENTATION FOR REAL-TIME H264/AVC VIDEO ENCODER ON MULTI-CORE DSP TMS320C6472

Nejmeddine Bahri[(1)], Thierry Grandpierre[(1)], Med Ali Ben Ayed[(2)], Nouri Masmoudi[(2)], Mohamed Akil[(1)]

(1) ESIEE Engineering / LIGM Laboratory, University Paris-EST-France
(2) National School of Engineers / LETI Laboratory, University of Sfax-Tunisia
nejmeddine.bahri@esiee.fr thierry.grandpierre@esiee.fr

ABSTRACT

In this paper, we exploit the parallelism offered by six-cores Digital Signal Processor (DSP) TMS320C6472 to implement the H264/AVC video encoder in order to meet the real-time constraint for different video resolutions. To enhance the encoding speed, GOP Level Parallelism approach is implemented on 5 slave DSP cores. A master core is reserved to manage data transfers among DSP memory and personal computer in order to perform a real-time video encoding demo taken into account video capture and bit-stream storage. Multithreading algorithm and ping pong buffers technique are used in order to optimize the communication overhead. Experimental results show that our enhanced implementation allows to overcome the real-time constraint by reaching up to 120 f/s (frame/second) for Common Intermediate Format resolution (CIF 352x288) and 35f/s for Standard Definition (SD 720x480). Our proposed approach can save about 80% of run-time for High Definition resolution (HD 1280x720). The Enhanced GOP Level parallelism approach on five DSP cores achieves a good average speedup factor of 4.88 without inducing any quality degradation or bit-rate increment.

1. INTRODUCTION

In our days, embedded systems occupy more and more our daily life. They invade many applications such as telecommunication, medical, defense and TV video coding etc. Actually, new embedded systems technologies as multi-core and multi-processor architectures allow designers to develop more complex applications that require high processing capability. H.264/AVC [1] video encoder is one of those applications. It is characterized by a better video coding efficiency compared to previous standards. However, this efficiency is accompanied by a high computational complexity that requires a high-performance processing capability to satisfy real-time constraint (25 to 30 f/s). The trend towards high definitions makes it hard to achieve real-time encoding on embedded mono-core processors due to processor frequency limitation. As a result, using multi-core and parallel architectures will be imposed to recompense this deficiency and to reduce the run-time of H264/AVC encoder. Several works have been published taken into account the potential parallelism of H264/AVC encoder. Different partitioning techniques are discussed based on applying functional partitioning algorithms, data partitioning algorithms or both. Multi-core, multi- processor and multi-threading encoding systems have been suggested in many papers [2] to [6]. In this paper an implementation of H.264/AVC encoder on a multi-core DSP TMS320C6472 applying GOP Level Parallelism (GLP) algorithm is detailed. A real time video coding demo is presented taken into consideration image capture from a digital camera linked to our DSP platform using Ethernet connection, DSP encoding and bit-stream saving. Hiding communication overhead is also presented based on performing a multithreading algorithm and exploiting the standard ping pong buffers technique.

The remainder of this paper is structured as follows: next section presents and discusses the different partitioning methods and some related works on the parallel implementations of H264/AVC encoder. The architecture of our multi-core DSP TMS320C6472 is described in section 3. Section 4 highlights our enhanced implementation of GLP algorithm on five slave DSP cores and details the experimental results. Finally, section 5 concludes this paper with some perspectives.

2. H264/AVC ENCODER : PARTITIONING METHODS AND RELATED WORKS

H.264/AVC encoder baseline profile includes several modules such as intra prediction, inter prediction, integer cosine transform, quantification, entropy coding etc. This standard splits a video sequence into a hierarchical structure. The top level of this structure is the sequence that includes one or more groups of pictures (GOP). Each GOP consists of one or more frames and always starts with intra frame (I). The other frames are predicted frames (P). Finally, the frames are divided into one or more slices, subdivided themselves into macroblocks (MB) and blocks. According to functions organization and hierarchical sequence structure in H.264/AVC encoder, there are mainly two partitioning approaches:

Task-level parallelization (TLP): it decomposes the encoder into several steps, identify them into a different group of tasks equal to the number of threads available on the system and run these groups of tasks simultaneously as a pipeline. Several works have applied this method as [2]. This approach ensure a low latency encoding but in the other side we can say that it is not suitable for H.264/AVC encoder because of the data dependencies between tasks that require a large amount of data transfers among processors; thus, consumption of the system bandwidth. Also functions in H.264/AVC encoder have not the same load balance which makes it hard to uniformly map

functions among processors. As a result, the final performance is always limited by the heaviest load processor.

Data-level parallelization (DLP): it exploits the hierarchical data structure of H264/AVC encoder by simultaneously processing several data levels on multiple processing units. DLP is restricted by data dependencies between different data units. We can note that no dependencies existed among different GOPs because each GOP starts with an intra frame. Hence, several GOPs could be encoded in parallel. This approach is called "GOP Level Parallelism". It has been adopted by several researchers as in [3].This method ensures the best encoding speedup but requires a large memory amount. Thus, it is not adequate for System on Chip platforms (SOC). Motion estimation in the H264/AVC encoder imposes a partial dependency between successive frames of the same GOP. Thus, multiple frames can also be encoded in a pipeline structure once the search window in the reference frame has been encoded. Consequently, this method is called "Frame Level Parallelism" [4]. It provides a compromise between encoding latency and implementation efficiency. Other works apply slice level parallelism such as in [5]. They split the frame into independent slices and simultaneously process them on different units. The major drawback of slice level parallelism is that it induces a bit-rate increment because some data dependencies are not respected. Finally, in the same frame, several MBs could be encoded at the same time once neighboring MBs of the current MB are encoded in order to respect spatial data dependencies. This scheme is called "MB level Parallelism" [6]. Large amount of data transfers and synchronizations between processors in addition to the non-equal load balance make the MB level parallelism approach not efficient for parallel H264/AVC video encoder. Despite all the different techniques and platforms used in order to accelerate the encoding speed, most of them have not succeeded to meet the real-time constraint (30 f/s) even for low resolution. For that, we try in this work to use a powerful platform and apply an efficient partitioning method in order to meet the real time video encoding constraint.

3. DSP PLATFORM DESCRIPTION

Software flexibility, low power consumption, time-to-market reduction, and low cost make DSPs an attractive solution for embedded systems implementations and high performance applications. Motivated by these merits and encouraged by the great evolution of DSP architectures, we chose to implement the H264/AVC encoder on the multi-core DSP TMS320C6472 [7] to profit from high processing frequency and an optimized architecture in order to achieve real-time embedded video encoder. Low power consumption and a competitive price tag make the TMS320C6472 DSP ideal for high-performance applications and suitable for many embedded implementations. Six C64x + DSP cores, very long instruction word (VLIW) architecture, 4.8 M-Byte (MB) of memory on chip, Single Instruction Multiple Data (SIMD) set and a frequency of 700 MHz for each core are combined to deliver 33600 MIPS performance. Each C64x+ core integrates a large amount of on-chip memory organized as a two-level memory system: 32 K-Bytes (KB) of L1P and L1D cache memories and 608 KB of local L2 memory. L2 memory can be configured as mapped RAM, cache, or some combination of the two. In addition to L1 and L2 memories dedicated to each core, the six cores share 768 KB of shared L2 memory. It is managed by a separate controller and can be configured as either program or data memory. This large quantity of on-chip memory can eliminate access to external DDR2 memory (256 MB), therefore reducing the power dissipation and accelerating algorithms processing since internal memory is faster than external memory. Performance is also enhanced by an EDMA controller that is able to manage memory transfers independently from the CPU. TMS320C6472 DSP supports different communication peripherals as Gigabit Ethernet for Internet Protocol (IP) networks, UTOPIA 2 for telecommunications and Serial RapidIO for DSP-to-DSP communications.

4. ENHANCED GOP LEVEL PARALLELISM IMPLEMENTATION

In this paper, we choose to apply the GOP Level parallelism approach in order to accelerate the encoding speed. Our choice is based on several reasons: First, this scheme ensures a good speedup factor without inducing any rate distortion (Quality degradation or bit-rate increase). Second, no dependencies between GOPs make this approach easy for implementation. No data transfers or synchronizations among processors are required. Finally, no memory constraint is required unlike SOC platforms. Our DSP includes enough memory space that is able to handle all the GOPs frames.

4.1 Video encoder demo

To ensure real time video encoding demo, we have to guarantee a real time frame reception by the DSP. For that, 332 MBits/s bandwidth (42 Mbytes/s) at least is required to transfer 30 frames/s HD 720p resolution on YUV 4:2:0 format ((1280x720x1.5)x8bits x30f/s). Since our DSP evaluation board has not yet any simple frame grabber interface, we use as fist step a personal computer (PC) linked to a Universal Serial Bus (USB) HD webcam to send the raw images to the DSP. Our DSP board and the PC support both a Gigabit Ethernet interface (1000 Mbits/s) that allows us to ensure a real time data transfers among them. The PC could be thereafter replaced by another embedded platform including camera interface and a Gigabit Ethernet communication peripheral.

As our platform includes 6 DSP cores, we exploited the first core "core0" as master. It is considered as a TCP server (transmission Control Protocol). It is devoted to establish TCP/IP (Internet Protocol) connection with the client (PC) exploiting Network Developer's Kit library [8]. It is used firstly to receive the GOPs sent by the camera board side after frames capture and save them into the

external memory which is a shared memory between all the DSP cores. Then, the 5 remaining DSP cores are considered as slaves and they are used to encode the 5 received GOPs.

Once the encoding is achieved, the core0 will send the bit-stream of all encoded frames to the PC in order to be stored in a file or decoded later. For each slave core (1 to 5) as shown on Figure 1, a memory section is reserved. It contains the GOP current frames, the reconstructed frame (RECT) and finally the bit-stream buffers where the bit-stream of each frame from the GOP will be saved.

4.2 Optimized GOP level implementation

To enhance the classic GOP level parallelism approach, we focused on hiding communication overhead. Our optimization is based on two strategies as shown on Figure 1: the first is using the ping pong buffers technique on the DSP side in order to overlap GOPs encoding process with reading and writing GOPs processes. The second is exploiting the multi-threading approach on the camera board side. Thus, three threads are created to handle: 1) Reading raw frames and sending them to DSP via Ethernet. 2) Receiving bit-streams from DSP. 3) Saving the received bit-streams in a file.

On the DSP side, for each slave core a ping pong GOP buffer is allocated for the current frames and also a ping pong bit-stream buffer is allocated for the generated bit-streams for each frame from the GOP. We keep a single buffer for the reconstructed frame because no transfers are required for this data. As a result, for each core, one buffer for the reconstructed frame, two GOP size buffers for the current frames and two GOP size buffers also for the bit-streams are allocated into the memory section for each slave core in SDRAM memory. In our work, GOP size is equal to 8. So, 16 buffers (8*2) for current frames, 16 buffers for bit-streams and one buffer for the reconstructed frame are allocated for each DSP core.

In the internal memory of core0, a TCP server program is loaded to establish Ethernet connection between the DSP and the PC. Our H264/AVC program is loaded into each internal memory of the 5 remaining cores. Local variables used during encoding such as predicted MB buffers, transform and quantification matrixes, best predicted modes etc are also allocated into the internal memory of each core to avoid data overlap among different cores.

A C/C++ project is implemented on the camera board side in order to capture raw frames from camera using OpenCv library [9] that allows also to convert captured frames from RGB to YCrCb 4:2:0 format that is used in our H264/AVC encoder. Finally, a TCP socket is created to transfer data between core0 and the camera board via the Gigabit Ethernet link.

The strategy of our implementation is described in Figure 2 and consists of the following steps:

- The first thread "thread1" captures the first frame from a camera or a file and sends it to core0 which will save it into the ping buffer SRC[0][0] of core1. Core0 sends then an inter processor communication interruption (IPC) to core1 to indicate that it can start encoding its current frame.
- When receiving IPC from core0, core1 starts encoding the first frame of the GOP and at the same time, thread1 continues reading the next frames of the first GOP and sending them to core0 which will save them into the ping buffers of core1 SRC1[0][i] (i=1 to GOP size-1).

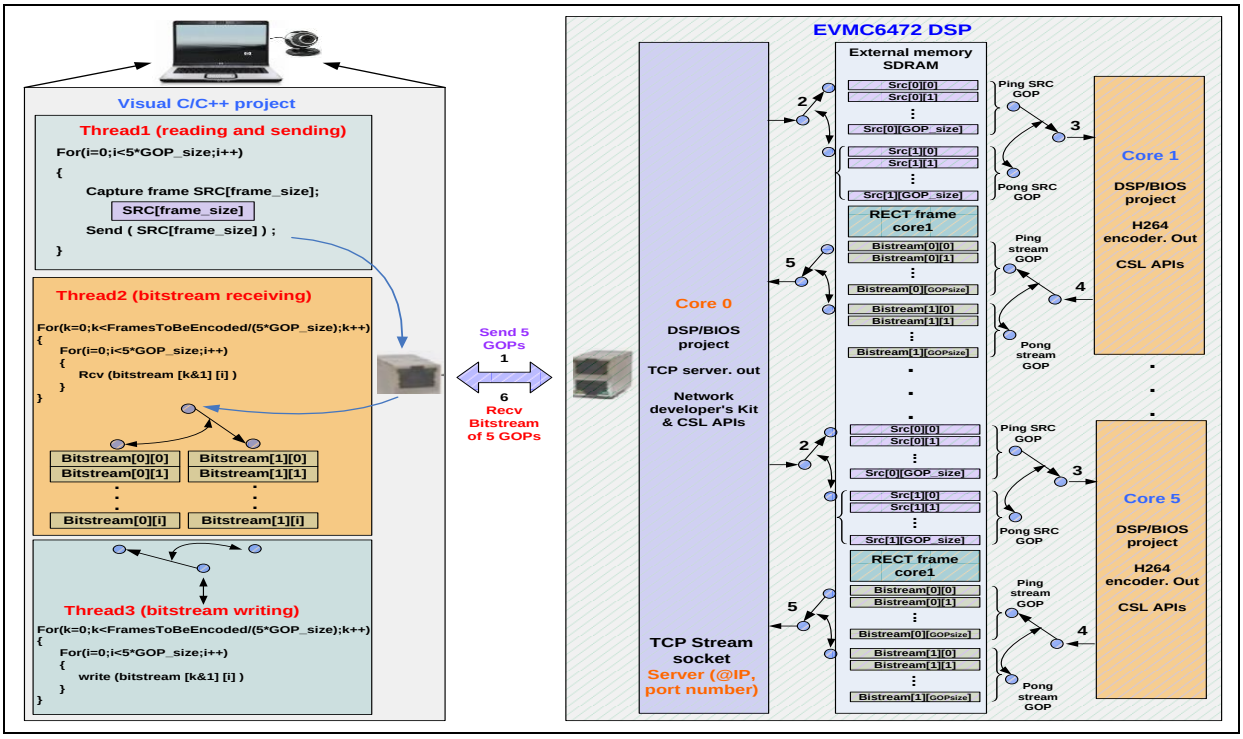

Figure 1 –Diagram Description of video encoding demo using enhanced GOP Level Parallelism approach

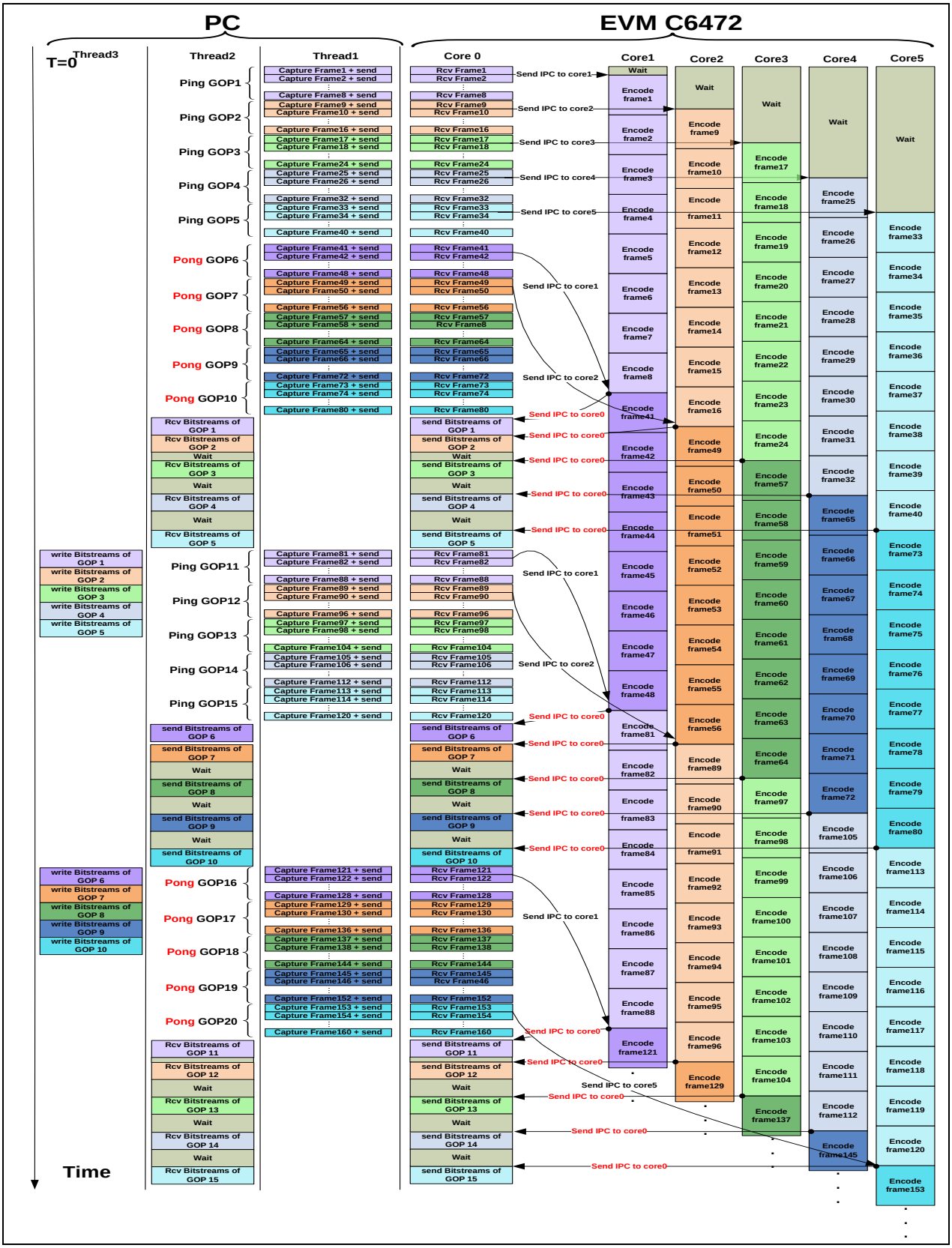

Figure 2 –The chronological steps of Enhanced GOP Level Parallelism on the multi-core DSP TMS320C6472

- When finishing reading and sending the first GOP, thread1 starts reading the second GOP and sends it to core0 which will save it into the ping buffers of core2 SRC2[0][i]. The same thing as the first GOP, when receiving the first frame of the second GOP, core0 sends an IPC to core2 to notify it that it can

start encoding the first frame of its GOP. This step is repeated until finishing the reception of 5 GOPs. So each core starts encoding as soon as the first frame of its corresponding GOP is received unlike the classic GOP level implementation where encoding is started after receiving all the frames of 5 GOPs.

- During encoding the first 5 GOPs by core1 to core5, thread1 sends the next 5 GOPs to core0 which will save them into the pong buffers SRC[1][i] for each core(i=0 to GOP size-1). Because encoding process takes more time than reading process, communication delays are hidden and they will not be added to the parallel run-time.
- When a DSP core finishes encoding its ping GOP and the bit-stream is saved into the ping buffers Bistream[0][i], it sends an IPC to core0 to notify it that it can send its bit-stream to PC. After that, this core starts encoding its pong GOP already received and saved into the pong buffers SRC[1][i] in its memory section without any wait. Then, it will save the bit-stream into the pong buffers Bistream[1][i] to not overwrite data stored into the ping buffers Bistream[0][i] which are being transferred by core0 to PC.
- Core0 sends the pings Bistream[0][i] to PC, starting with the bit-streams of core1 and finishing with the bit-streams of core5 in order to be saved in a chronological order. The second thread "thread2" receives the ping bit-steams and saves them into the ping buffers Bistream[0][i]. After that, the third thread "thread3" writes the bit-streams in a file and at the same time thread1 sends the next 5 GOPs to core0 which will save them into the ping buffers SRC[0][i] of each core. With this technique, ping bit-streams writing, pong SRC frames encoding and capturing the next 5 ping GOPs are processed in parallel.
- The work is then reprocessed in a reverse order for SRC frames and bit-streams through ping pong buffers.

4.3 Cache coherency

Multi-core processing often involves cache coherency problem. This is returns to the simultaneous access of two or more cores with a separate cache memory to the same location in a shared memory. In general purpose multiprocessor, programmers don't have such problem because it is controlled automatically by a complex hardware. In our multi-core DSP architecture, designers have to manage cache memory, since there is no such automatic controller. In order to deal with cache coherency, the Chip Support Library (CSL library) [10] from TI provides two API commands:

- CACHE_wbL2((void *)XmtBuf, bytecount, CACHE_WAIT) to write back the cached data from the cache memory to its location in the shared memory.
- CACHE_invL2((void *)RcvBuf, bytecount, CACHE_WAIT) to invalidate the cache lines and force the CPU to read data from its location in the shared memory.

In our implementation, after reading the captured frames from PC, core0 should write back the cached data to their locations in external memory in order to be used later by the remaining cores for encoding. In the other side, cores 1 to core5 should invalidate the current frames addresses in the cache memory before starting encoding in order to encode the updated data written by core0 and not the old data existed in their cache memories. Moreover, after finishing encoding, cores 1 to core5 should write back the bit-streams from cache memory to external memory and similarly core0 should invalidate the bit-streams in its cache memory in order to send the new values to PC.

4.4 Experimental results

Experiments are performed on several video sequences with different characteristics and resolutions: CIF, SD and HD on 5 DSP cores running each at 700MHz. The used GOP size is 8 and the number of encoded frames is 300 for CIF resolution and 1200 frames for SD and HD resolutions. For performance evaluation, encoding speed is computed for mono-core and multi-core implementation using GOP level parallelism approach on 5 slave DSP cores.

In our tests, data transfer time which consists of frames capturing, GOP structure transferring to DSP, receiving them by core0, and loading them to DSP memory is included in our calculation and added to the encoding time in order to evaluate our enhancement techniques for hiding communication overhead.

TABLE I
ENCODING SPEED FOR CIF (352X288) RESOLUTION FOR MONO-CORE AND MULTI-CORE IMPLEMENTATIONS

CIF sequence	Encoding speed on a single core (f/s)	Encoding speed on 5 DSP core (f/s)	Speedup
Foreman	24.90	121.74	4.89
Akiyo	25.56	123.32	4.82
News	26.03	127.76	4.91
Container	25.68	125.37	4.88
Tb420	23.37	114.12	4.88
Mobile	22.42	109.16	4.87
average	**24.66**	**120.24**	**4.88**

TABLE II
ENCODING SPEED FOR SD (720X480) RESOLUTION FOR MONO-CORE AND MULTI-CORE IMPLEMENTATIONS

SD sequence	Encoding speed on a single core (f/s)	Encoding speed on 5 DSP core (f/s)	Speedup
Planets	7.24	35.23	4.87
Power of natures	7.19	35.28	4.91
Tortue	7.29	35.51	4.87
Vague	7.13	34.65	4.86
Nature	7.36	36.29	4.93
Bird	7.93	38.34	4.83
average	**7.36**	**35.88**	**4.88**

TABLE III
ENCODING SPEED FOR HD (1280X720) RESOLUTION FOR MONO-CORE AND MULTI-CORE IMPLEMENTATIONS

HD sequence	Encoding speed on a single core (f/s)	Encoding speed on 5 DSP core (f/s)	Speedup
Planets	2.79	13.72	4.92
Power of natures	2.74	13.27	4.84
Tortue	2.78	13.63	4.90
Vague	2.79	13.52	4.85
Nature	2.81	13.79	4.91
Bird	3.03	14.76	4.87
average	**2.82**	**13.78**	**4.88**

Table I, II and III show respectively the encoding speeds and speedups for CIF, SD and HD resolutions for the H264/AVC encoder on a single core and on a 5 DSP cores. Experimental results show that mono-core implementation does not meet the real-time constraint (30f/s) even for low resolution. Applying our enhanced multi-core implementation on 5 DSP cores allows us to overcome the real-time encoding constraint by reaching up to 120 f/s and 35 f/s in average for CIF and SD resolutions respectively. For HD resolution, real-time is unfortunately not achieved but encoding speed is efficiently improved. Run-time's gain might reach up to 80%. Regarding rate distortion, our implementation does not induce any visual quality degradation or bit-rate increase since data dependencies are respected.

Applying multi-core implementation on 5 DSP cores allows getting an interesting encoding speedup by a factor of 4.88 in average for the different resolutions. This speedup factor is very close from the theoretical value which is 5. This tiny decrease in speedup factor is first due to inter-communications needed among the master (core0) and the slaves (core1 to core5) such as write-backs and cached data invalidations and second to the impossibility of simultaneous access to SDRAM memory by all DSP cores to read and write data.

Although the data transfer time is taken into account, this time does not affect the encoding speed which affirms that our proposed data transfer scheduling technique completely hides the communication overhead. In fact, data transfer times will be only noticed only for the first GOPs as shown on figure 2 but after that, these transfers are overlapped with the encoding process. So, when testing an importing number of frames, the first transfer time will be insignificant and does not induce a performance penalty especially if we know that encoding process is more important than reading or writing processes.

5. CONCLUSION

In this paper, an optimized implementation of H264/AVC encoder on a multi-core DSP TMS320C6472 was presented. GOP Level parallelism approach was applied to accelerate encoding speed. Exploiting a multi-threading algorithm combined with using a ping pong buffers technique allows enhancing our GOP multi-core implementation and efficiently hides communication overhead. Experimental results for the enhanced GOP level parallelism on 5 DSP cores running at 700 MHz showed that real-time was achieved by attaining up to 120 f/s and 35 f/s in average as encoding speeds respectively for CIF and SD resolutions. Our parallel implementation saved about 80% of run-time for HD resolution and ensured a good encoding speedup factor of 4.88 without resulting any video quality degradation or bit-rate increase. As perspectives, we will try to achieve a real-time encoding for HD resolution by implementing our approach on the latest generation of Texas Instruments DSP (TMS320C6678) which includes 8 DSP cores each running at 1.25 GHz, giving a large possibility to meet the real-time constraint. Also, we will move to the implementation of the new video standard HEVC-H265 on the same DSP exploiting our knowledge on multi-core DSP implementation. Finally, it is important also to notice that our demo architecture based on a camera board connected to a multi-core DSP board can be used in a lot of image and video processing applications.

ACKNOWLEDGEMENT

This work is sponsored by the French ministries of Foreign Affairs and Tunisian ministry for Higher Education and Scientific Research in the context of Hubert Curien Partnership (PHC UTIQUE) under the CMCU project number 12G1108.

REFERENCES

[1] Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, "Draft ITU-T Recommendation and Final Draft international Standard of Joint Video Specification (ITU-T Rec. H.264 ISO/IEC 14496-10 AVC)", JVT-G050, 2003.

[2] Zhibin Xiao, Stephen Le and Bevan Baas," A Fine-grained Parallel Implementation of a H.264/AVC Encoder on a 167-processor Computational Platform," ACSSC 2011 – Pacific Grove, CA, 2011.

[3] S.Sankaraiah, H.S.Lam, C.Eswaran and Junaidi Abdullah, "GOP Level Parallelism on H.264 Video Encoder for Multicore Architecture," International Conference on Circuits, System and Simulation IPCSIT vol.7 IACSIT Press, Singapore 2011.

[4] Zhuo Zhao; Ping Liang, "A Highly Efficient Parallel Algorithm for H.264 Video Encoder," Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on , vol.5, no., pp.V,V, 14-19 May 2006.

[5] António Rodrigues, Nuno Roma, and Leonel Sousa," p264: Open Platform for Designing Parallel H.264/AVC Video Encoders on Multi-Core Systems," NOSSDAV '10 Proceedings of the 20th international workshop on Network and operating systems support for digital audio and video Pages 81-86, Amsterdam, The Netherlands, 2010.

[6] Shenggang Chen; Shuming Chen; Huitao Gu; Hu Chen; Yaming Yin; Xiaowen Chen; Shuwei Sun; Sheng Liu; Yaohua Wang, "Mapping of H.264/AVC Encoder on a Hierarchical Chip Multicore DSP Platform," High Performance Computing and Communications (HPCC), 2010 12th IEEE International Conference on , vol., no., pp.465,470, 1-3 Sept. 2010

[7] TMS320C6472 datasheet, online available: http://www.ti.com/lit/ds/sprs612g/sprs612g.pdf

[8] TI Network Developer's Kit (NDK) v2.21 User's Guide, online available: http://www.ti.com/lit/ug/spru523h/spru523h.pdf

[9] Open source computer vision library: http://opencv.org/

TMS320C6472 Chip Support Library API reference Guide, online available: http://software-dl.ti.com/sdoemb/sdoemb_public_sw/csl/CSL_C6472/latest/index_FDS.html